IN THE UNITED STATES
PATENT AND TRADEMARK OFFICE

Attorney Docket No.: **Verizon-19/APP (02-1502)**

Appl. No.: **10/628,080**

Applicants: **Malathi VEERARAGHAVAN, Tim MOORS**

Filed: **July 25, 2003**

Title: **METHODS AND APPARATUS FOR AUTOMATING TESTING OF SIGNALLING TRANSFER POINTS**

TC/A.U.:

Examiner: **Not yet assigned**

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 C.F.R. §1.131

The undersigned Applicants, **Malathi Veeraraghavan** and **Tim Moors** hereby declare that the attached five page article titled "Experiences in automating the testing of SS7 Signalling Transfer Points" published in ACM SIGSOFT Software Engineering Notes, Volume 27, ISSUE 4 (July 2002) pages 154-158 ("the Attached Article") describes Applicants' own work.

Applicants further declare that they are the sole inventors of the subject matter to which the present patent application, S.N. **10/628,080,** is directed and the subject matter claimed in this patent application. The individuals named on the Attached Article, other than than the two named Applicants, were merely working under Applicants' direction and are not co-inventors.

Applicants further declare that the attached article was published either by Applicants or on Applicants' behalf.

The undersigned, hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Malathi Veeraraghavan Date Dec. 9, 03

Malathi Veeraraghavan

_____ Date _____

Tim Moors

Experiences in automating the testing of SS7 Signalling Transfer Points

Tim Moors[†], Malathi Veeraraghavan, Zhifeng Tao, Xuan Zheng, Ramesh Badri[‡]

[†] Uni. of New South Wales
Sydney, NSW 2052, Australia
+61 2 9385 4885
t.moors@unsw.edu.au

Polytechnic University
5 Metrotech Center
Brooklyn NY 11201 USA
+1 718 260 3050
mv@poly.edu
jefftao@photon.poly.edu
zhxfifa@photon.poly.edu

[‡] Sprint PCS
600 Cummings Park, Suite 2850
Woburn, MA 01801 USA
+1 781 638 1547
rbadri01@sprintspectrum.com

ABSTRACT

Signalling System 7 (SS7) is widely used for telephone signalling. Service providers need to frequently test their Signalling Transfer Points (STPs), which switch SS7 messages, for both protocol conformance and interoperability. This paper describes a system that automatically analyzes the data collected during STP tests. It consists of files that describe how the STPs are expected to behave during the test, and Perl code that translates this Expected Behavior into a program that can search the data collected during the test for the expected events, and report on whether the system passed the test. The system readily processed over 30,000 events for each test run, and identified abnormal behavior that could interfere with interoperability and protocol conformance.

Categories and Subject Descriptors

B.4.5 [Input/Output and Data Communications]: Reliability, Testing, and Fault-Tolerance

C.2.2 [Computer-Communication Networks]: Network Protocols – *protocol verification*

General Terms

Design, Reliability, Experimentation, Verification.

Keywords

Automation, Signalling System 7, Signaling System 7, SS7, STP

1. INTRODUCTION

Signalling System 7 (SS7) [6] is used around the world to provide the signalling required for telephone communications. A core component of an SS7 network is the Signalling Transfer Point (STP). Service Switching Points (SSPs) generate SS7 messages, and act as their ultimate destinations, while STPs switch these messages on their path from origin to destination. The STPs are deployed in mated pairs, so that they can maintain the switching service even when one STP fails. A mated pair of STPs may connect to a set of SSPs, to Service Control Points (SCPs) that provide database functions, or to another mated pair of STPs, forming a "quad" configuration, as shown in Figure 1.

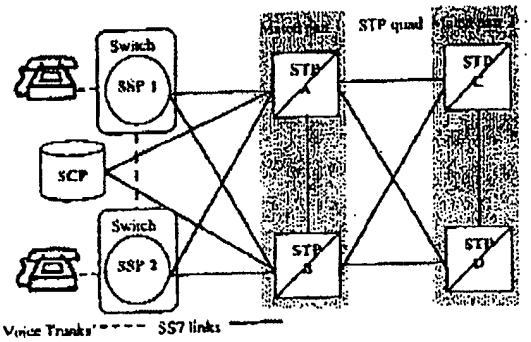

Figure 1: Main elements of an SS7 network.

Pairs of STPs that are interconnected to form a quad may belong to an SS7 network operated by a single organization, or may form the interface between SS7 networks from different organizations. Interfaces between SS7 networks allow a Local Exchange Carrier to send its signalling traffic over another carrier's network. This is becoming increasingly important in the United States as regulatory bodies are encouraging competition between Local Exchange Carriers by requiring incumbent carriers to open their network to the signalling traffic of competitors, in exchange for being granted rights to provide long-distance services. However, different carriers may source their STPs from different vendors or configure their STPs differently, and this leads to a need to test the interoperability of STPs from different carriers. A single carrier may even source STPs from different vendors, for reasons such as risk management or encouraging competition between vendors for the supply of equipment. Thus, carriers have a strong need for interoperability [1] and conformance [5] testing of STPs.

For a carrier to ensure that the signalling network continues to operate correctly, it needs to regularly test STP quad interoperability. A carrier will often deal with STPs from three or more different vendors, and each vendor regularly issues new releases of the software and hardware for their STPs, e.g. multiple

times per year, and the carrier may interface with other carriers who reconfigure their STPs. Not only must the testing be performed regularly, but it is complicated (due to the nature of the SS7 protocol), time consuming, and (for the most part) mundane. For example, by one estimate, the analysis of each test takes 933 hours of labor, and over five years the test analyses would cost approximately $1.5M. These characteristics make STP quad interoperability testing a good candidate for automation.

Verizon Communications, who operate one of the world's largest SS7 networks in support of their US operations, recently decided to automate their STP quad interoperability testing. This automation project divided the testing process into four stages: setting up the test environment, executing the tests, retrieving data collected during the test, and analyzing the test data. Verizon sponsored Polytechnic University to create and implement a system for the analysis stage. This system is called the Automated SS7 Test Result Analyzer (ASTRA). This paper focuses on ASTRA, and the analysis stage of the testing process. While ASTRA was designed for use in conjunction with other stages of testing that are automated, it has also been successfully used to analyze the results of tests that were conducted manually.

Protocol testing [8] requires a Test Plan (e.g. [4]) that specifies the stimuli that will be applied to the System Under Test (SUT) with the intention of exercising its functionality. From this Test Plan, and knowledge of STP requirements [2] and the operation of the SS7 protocol [3], we developed a description of the Expected Behavior (EB) of the STPs for each test in the plan. The analysis process starts with ASTRA taking as input the data collected during the test, a description of the network configuration used during the test, and a record of when each test step was performed, and formatting these into an Actual Event Record (AER). ASTRA then compares this concrete AER with the abstract EB, and classifies events described in the AER and EB as being either matched (in EB and AER), missing (in EB but not AER), hidden (in EB but not in AER and not anticipated due to monitoring limitations), or unexpected (in AER but not in EB). ASTRA then statistically analyzes the events observed in the AER in order to infer STP parameters, and presents these and information about the test events in a report that describes how the STPs failed during the test. Figure 2 shows this architecture.

The structure of this paper follows that of ASTRA: Section 2 describes the format used to represent the EB of the SUT, and Section 3 describes translation of the EB into an Analyzer. Section 4 describes the formatting of test data, and Section 5 describes how events in the EB are matched with those in the AER. Section 6 describes some bugs that ASTRA detected when analyzing data from one test run. Section 7 concludes the paper. An extended paper [7] describes ASTRA in more detail.

2. THE EXPECTED BEHAVIOR (EB)

This section describes the EB Expression Language (EBEL) that captures how we expect STPs to behave during a test.

A key feature of the EB is that one event can cause multiple succeeding events, and the relative order of these succeeding events is not always predictable. For example, referring to Figure 1, the failure of the link connecting SSP 1 to STP A should cause STP A to send Transfer Restricted (TFR) messages to neighboring SSP 2 and the SCP, indicating that it no longer

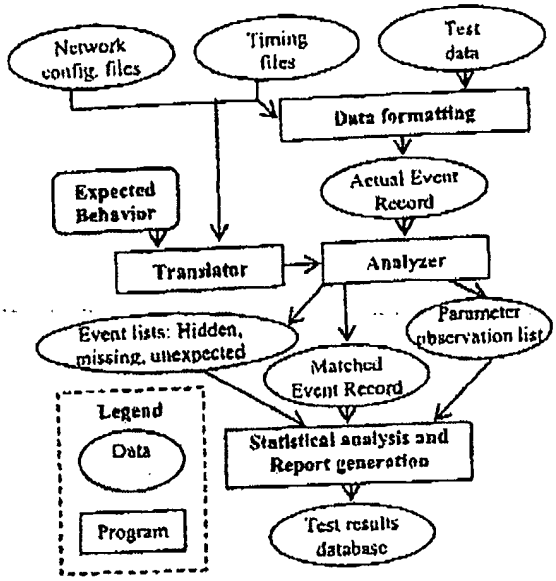

Figure 2: Architecture of the analysis system.

provides a direct path to the SSP whose link failed. The protocol does not specify the order in which STP A must send these two TFR messages. For our tests, each expected event is triggered by a single predecessor: there is no situation in which an event depends on multiple independent predecessors. Thus, the EB can be expressed as a tree, as shown in Figure 3.

The heart of the EB for a test is a numbered list of events that are expected during the test, with each event indicating the numbers of any other events that it causes. Figure 3 provides an abstract example of the tree structure of EBs, showing which events cause which other events, with the numbers corresponding to the line numbers of a particular EB shown in Figure 4, and letter designations indicating different iterations of a loop. To describe the EB, we created text files that described one event on each line, with lines describing events that caused other events containing the phrase "`causes`" followed by a list of the line number(s) of the event(s) that the event on this line causes. To manage the numbering and references, we created the EB using Microsoft Word, using its line-numbering, cross-references and update features, and saved these as text files for automated processing.

The EB lists stimuli to the SUT, and the responses of the SUT.

The chain of events that stimulate the SUT during the tests define the Test Plan, and the Generic Commands and control structures used to represent these events can be used in both the EB, and also in a Test Action List that drives automated test execution. The Generic Commands and corresponding stimuli are:

`fail_link()`, `restore_link()`: fail or restore a specific link in the network configuration,

`set_load()`: set the load on a particular link to a particular level, and

`pause()`: wait a certain period (e.g. to allow network traffic to stabilize) before injecting the next stimulus.

In our testing, we were able to express the Test Action List in an open-loop manner so that the test actions, and their timing, did

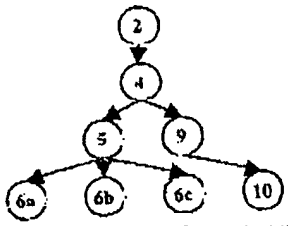

Figure 3: Abstract expected events form a tree of causation.

not depend on the behavior of the SUT. In the EB, each line containing a test action also contains a symbolic label (e.g. `$ta_fail{$s}`, indicating that the value of the variable `$s` should be appended to the string `$ta_fail`, to create a label for the test action on this line) that other parts of the EB (e.g. `while` loops) can use to refer to the time of the test action. The matching process uses these to define the scope of the search for an event. The EB uses symbolic names for these times, since it is produced before the tests are performed. After the test, during analysis, these symbols are instantiated with the actual times when each test action was performed.

The messages that an STP transmits on the links are the most common measure of the response of an STP, and the EB may be concerned with the type and parameters of a specific message, or with the aggregate volume of messages flowing over a particular path. The EB includes `findmsg()` commands to find a message with particular properties (e.g. type and originating and destination point codes) passing over a certain link or an arbitrary link, and `assert_load()` commands to check whether the load on a link is within a specified tolerance of a certain level. The STP's log of events that it observes is another important response, since accurate logs help alert network operators of imminent problems, and help problem diagnosis.

The parameters to `findmsg()` can be specified as wildcards (*), indicating that the matching process should ignore that parameter (e.g. a link, node or message type identifier) when trying to find a matching message. Wildcards are convenient in the EB to determine if an STP sends a particular message to *any* other node (wildcards for link name and destination), or whether an STP sends *any* type of message to a particular node.

The EB also includes control structures that allow iteration of the stimuli and responses, e.g. a `foreach` loop can be used with a `fail_link()` command, to fail each link connecting an STP to adjoining SSPs, and a `while` loop can be used with a `findmsg()` command, to indicate that an STP is expected to continue sending a certain type of message until a certain time.

A `waitfor()` command allows the EB to describe situations in which the SUT is expected to wait for a certain period before making some response. The first `waitfor()` argument provides the symbolic name of the parameter that specifies the period that the STP should wait for, and the second indicates which STP's parameter to use. Part of the matching stage (Section 5) involves recording the observed values of these parameters, so that the report generation stage [7] can infer the value of this parameter that the STP is using and compare it to the value that the STP claims to be using.

The `waitfor()` command is used in two different ways: Within `while` loops, it regulates the speed at which the while loop repeats, and directly causes only one other event, e.g., describing the expectation that an STP will send a certain message regularly while a link is unavailable. Outside of `while` loops, it is used to indicate a delay between one event and one or more consequent event(s), and so may cause multiple events.

Figure 4 provides a sample extract of an EB, showing how the STPs are expected to respond to the sequential failure of links connecting SSPs to STP A. In this example, `@SSPS_E` is a set of SSP identifiers, and the `foreach` loop is repeated to fail the link connecting STP_A to each SSP (`$s`) in this set. The set `@set_1` describes the nodes (SSPs and SCPs) that remain connected to STP_A after the link has been failed; the extract of the code does not show where `@set_1` is updated. In this example, a period T11 after STP A detects the link failure, it is expected to send TFR messages to STP D and the remaining neighboring SSPs. Once STP D receives the TFR message, it should repeatedly (with period T10) send a Signalling Route-Set-Test-Restricted (RSR) message to STP A enquiring if the path has been restored. `$time` is a global variable that is updated during the matching process to indicate the time of the latest actual event that the matching process has attempted to match with an expected event.

Creating the EB for each test in the Test Plan, and validating our expectations against observed behavior, proved to be a time-consuming task. For the twelve tests in our Test Plan, the EB totalled almost 6,000 lines and described over 30,000 events.

3. THE TRANSLATOR

Apart from the "causes" construct and line numbering, the language described thus far for the EB bears a striking resemblance to Perl [9], with the addition of calls to functions such as `findmsg()` and `waitfor()`. Originally, we intended to write a Perl script that interpreted the EB to determine what events were expected next and then search the AER for those events. But then we realized that we could simply translate the EB file itself into a Perl script: If one event causes a group of other events, then this can be expressed in Perl as the first event being the condition of an `if` statement, and the consequent events being placed in the body of the `if` statement. When an event causes a group of events surrounded by a control structure (e.g. event 4 causes a `foreach` loop), then that control structure is included in the body of the `if` statement that depends on that event. Figure 5 shows the nesting of `if` statements to represent the abstract event tree of Figure 3. Each number of Figure 5 can be replaced with the line from Figure 4 that has the same number, creating the hierarchy of `if` statements corresponding to Figure 3.

The depth of the tree of expected events can lead to many levels of hierarchy in the `if` statements, producing code that is difficult for humans to read. That is acceptable because the translated EB is only used by the automated system, and is precisely why we did not originally express the EB as a hierarchy of `if` statements.

The hierarchy of `if` statements leads to a program that performs a depth-first search of the tree of expected events. The search for an event covers all actual events that occur between the times of the antecedent event (stored in a variable `$time`) and the time of the next test action (stored in another variable `$next_action_time`). Events that are enclosed in a `while`

```
1. foreach $s (@SSPS_E) {
2.   $sta_fail{$s}: fail_link(A(STP_A)($s)) causes 4,...
3. ...
4.   waitfor(T11,STP_A) causes 5,9,...
5.   foreach $t (@set_1) {
6.     findmsg(A(STP_A)($s),$t,STP_A,$t,TFR,$s)
7.   }
8. ...
9.   findmsg(D(STP_A)(STP_D),STP_D,STP_A,STP_D,TFR,$s)causes 10
10.  while($time<$sta_restore{$s}) {
11.    waitfor(T10,STP_D) causes 12
12.    findmsg(D(STP_A)(STP_D),STP_A,STP_D,STP_A,RSR,$s)
13.  }
14. ...
15.foreach $s (@SSPS_E) {
16.  $sta_restore{$s}: restore_link(A(STP_A)($s)) causes ...
```

Figure 4: Extract of the EB of an STP (with ellipsis added and lines renumbered).

```
if( 2 ) {
  if( 4 ) {
    /*5*/ foreach $t (@set_1){
          6$t
    }
  }
  if( 9 ){
    10
  }
}
```

Figure 5: Representing the abstract event tree of Figure 3 as a series of nested if statements.

loop form an exception to this general rule; their search continues until the time specified as a parameter to the `while` loop, not until the time of the next test action.

ASTRA includes a Translator, written in Perl, that translates the EB into an Analyzer. Translation is similar to the compilation of a program written in a high-level language such as C: It takes as input a program that is in human-readable form, and produces a program that can be executed. In the case of ASTRA, the Analyzer that results from translation is a Perl program. The translation also links the Analyzer to libraries of functions that are used during the matching process, and for formatting results.

The Translator is run after a test is run so that the Analyzer can include information that is specific to the test run. This information includes the network configuration (which affects the definitions of sets of nodes used in the EB), the list of links that were monitored during the test (which determines which events will be classified as being "hidden"), and the times of test actions (which affect the searching operation).

A secondary role of the Translator is to help people create EB files, by checking the syntax of an EB file, and converting identifiers in the EB into the form required by Perl: enclosing names of links and nodes in quotes (e.g. `STP_B` ? `"STP_B"`) and prefacing function calls with ampersands and suffixing them with semicolons (e.g. `findmsg(...)` ? `&findmsg(...);`).

4. DATA FORMATTING

The data formatting is fairly mundane, but nonetheless necessary. It ensures consistency of data from all monitors, adjusting for variations in format from different monitor types, and variations in synchronization of different monitors. It integrates the records from multiple monitors to form a single AER, and filters the AER to remove irrelevant information that may be introduced by STPs that are configured with information (point codes) about nodes that were not involved in the test (to avoid time-consuming precise configuration). It abstracts the numerical form of identifiers used by monitoring equipment (e.g. point code 246-022-001) into the human-friendly symbolic form used in the EB, so the EB and AER are comparable. Finally, it sorts the events in the AER into chronological order, so as to expedite searching during event matching, and adds to each event a "matched" field that is used by the matching process.

It is worth pointing out two practical realities of our experience with monitoring equipment: The high cost and limited availability of monitoring equipment prevented comprehensive monitoring of all points in the test network. Consequently, one input to ASTRA is a list of monitoring points for which data was not collected. When an event occurs in the EB, but the corresponding point is not monitored, then ASTRA classifies the event as being "hidden". Also, deviations between the EB and AER may be due to failure of the monitoring equipment, not of the SUT. Amongst the vast volume of data that we analyzed, we noticed a couple of instances in which the monitoring equipment seemed to provide incorrect reports (e.g. missing a digit in a timestamp) or was just as suspicious as the STPs under test (e.g. the interval between periodic messages would occasionally be twice the normal value, suggesting that an interstitial message was dropped by either the STP or monitoring equipment).

5. EVENT MATCHING

The goal of event matching is to classify events in the EB and AER as being either hidden, missing, matched or unexpected. Figure 6 shows and how these event classifications relate to which events were expected, which actually occurred, and which were observable given the monitors used during the test. An event's classification is recorded by writing it to either the hidden, missing, matched or unexpected event list.

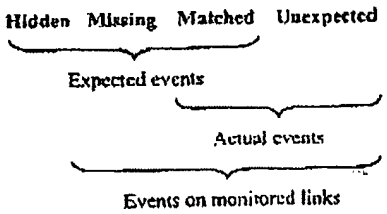

Figure 6: Relationship between the four event classifications and sets of expected and actual events, and monitor coverage.

The first stage of event matching is driven by the EB: The analyzer checks if each event in the EB was expected on a link that was monitored during the test. If not, then the event is classified as hidden. If the link was monitored, then the analyzer searches the AER for the event. Events that can't be found in the AER are classified as missing, while others are classified as being

matched. If the matched event causes other events, then ASTRA also searches for those events from the time of the matched event. Before elaborating on the details of the first stage, we will briefly describe the second stage:

The second stage of event matching is driven by the AER, or more precisely the Matched Event Record (MER), which indicates which events in the AER were matched to the EB, and which were not. This second stage takes each event in the MER that was *not* matched to the EB, and classifies it as unexpected, writing it to the Unexpected Event List.

There are three important details about the operation of the first stage of event matching. These relate to the search depth, the recording of parameter observations, and the matching of events that use wildcards, and are discussed in the following paragraphs.

Search depth: Whenever ASTRA fails to match an event, it prunes the tree of expected events at that event; i.e. it does not search for consequent events that the missing event was expected to cause. If the consequent events were also missing, then they will *not* appear in the Missing Event List, since the omission of their antecedent implies that they will be missing. This reduces the list of missing events so that a human analyst can concentrate on the most events that were missing, and need not be inundated with a mass of consequent events that may be missing. If the consequent events were not missing, e.g. if monitoring equipment simply failed to detect the antecedent event, then they will appear in the AER, and ASTRA will mark these as being unexpected. Thus, a single missing event can cause multiple consequent events to be marked as unexpected.

Parameter observations: When ASTRA matches an event that is caused by a `waitfor()` statement, it records an observation of the delay between the event that caused the `waitfor()` statement, and the event that the `waitfor()` statement caused. This observation measures an instance of the parameter of the `waitfor()` statement that indicates how long the STP should wait before causing the next event. The report generation stage [7] provides a statistical analysis of these parameter observations, allowing an analyst to infer STP parameter values. The details of parameter observations depend on whether or not the `waitfor()` statement was included within a `while` loop.

Observations that indicate missing events: ASTRA only marks an event that is caused by a `waitfor()` statement in a `while` loop as being missing if it was *never* found. Other missing events are detected by the distribution of the parameter observations. For example, if an STP was expected to send a message every T10 seconds, but every second message the STP tried to send was lost, then ASTRA would indicate that the message was found, but that the timer parameter was observed to be 2×T10 seconds.

6. TEST RESULTS

When we ran ASTRA on the data produced during a series of interoperability tests between STPs from two manufacturers, we detected three bugs with the STPs from one manufacturer that could interfere with interoperability. These bugs appeared in a link failure test. According to the SS7 protocol, an STP should respond to link failure or restoral by sending a message to neighboring STPs indicating that traffic flow through it is either *restricted* (because it has poor connectivity to the destination), *prohibited* (because it has no connectivity to the destination), or *allowed* (because a link has become available, enabling connectivity to the destination). In our tests, the STPs from one vendor sometimes responded to link failure by indicating that traffic was *allowed* to flow, rather than it was indicating that it was restricted or prohibited. These STPs also sometimes sent restriction messages, when they were expected to send prohibition messages, allowing a routing loop to occur. Finally, these STPs were sometimes unnecessarily conservative in that they indicated to their mated pair that traffic was prohibited when it should only have been restricted. ASTRA readily allowed the human analyst to pinpoint this unexpected behavior amidst the 30,000 other events that occurred during the testing.

7. CONCLUSION

ASTRA is able to automate the analysis of data produced when testing communication protocols. While ASTRA was designed to solve the specific problem of analyzing the results of SS7 tests, its underlying mechanism (e.g. language for expressing the EB, process of translating an EB into a Perl script, and the event matching process) could also be applied more generally to analysis of tests of other protocols.

8. ACKNOWLEDGMENTS

We would like to thank Verizon for sponsoring this project, and for their help through the following staff: N. Cocker, R. Iantel, M. Lin, J. Murphy, G. Ormazabal, W. Peer, J. Sylvester, and J. Vecchioli. The New York State office of Science, Technology and Academic Research also sponsored this project through the Center for Advanced Technology in Telecommunications at Polytechnic University. Thanks also to W. Preedalumpabut.

9. REFERENCES

[1] J. Alilovic-Curgus and S. Vyong: "A framework for interoperability testing of network protocols", *Proc. Int'l Conf. on Network Protocols*, pp. 376-83, 1993

[2] Bellcore: "Signaling Transfer Point (STP) Generic Requirements", Generic Requirement GR-82-CORE; Issue 2, Dec. 1996

[3] Bellcore: "Specification of Signalling System Number 7", Generic Requirement GR-246-CORE; Issue 3, Dec. 1998

[4] ITU-T: "Signalling System No. 7 Protocol Tests", ITU, Recommendation Q.755, Mar. 1993

[5] S. Kang: "Relating interoperability testing with conformance testing", *Proc. Globecom*, pp. 3768-73, 1998

[6] A. Modarressi and R. Skoog: "An overview of Signaling System No. 7", *Proc. IEEE*, 80(4):590-606, Apr. 1992

[7] http://www.ee.unsw.edu.au/~timm/pubs/02issta/long.pdf

[8] D. P. Sidhu and T.-K. Leung: "Formal methods for protocol testing: A detailed study", *IEEE Transactions on Software Engineering*, 15(4):413-26, 1989

[9] L. Wall and others: *Programming Perl*, 3rd ed., O'Reilly & Associates, 2000

IN THE UNITED STATES
PATENT AND TRADEMARK OFFICE

Attorney Docket No.: **Verizon-19/APP (02-1502)**

Appl. No.: **10/628,080**

Applicants: **Malathi VEERARAGHAVAN, Tim MOORS**

Filed: **July 25, 2003**

Title: **METHODS AND APPARATUS FOR AUTOMATING TESTING OF SIGNALLING TRANSFER POINTS**

TC/A.U.:

Examiner: **Not yet assigned**

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 C.F.R. §1.131

The undersigned Applicants, **Malathi Veeraraghavan** and **Tim Moors** hereby declare that the attached five page article titled "Experiences in automating the testing of SS7 Signalling Transfer Points" published in ACM SIGSOFT Software Engineering Notes, Volume 27, ISSUE 4 (July 2002) pages 154-158 ("the Attached Article") describes Applicants' own work.

Applicants further declare that they are the sole inventors of the subject matter to which the present patent application, S.N. **10/628,080**, is directed and the subject matter claimed in this patent application. The individuals named on the Attached Article, other than than the two named Applicants, were merely working under Applicants' direction and are not co-inventors.

Applicants further declare that the attached article was published either by Applicants or on Applicants' behalf.

The undersigned, hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

_____ Date _____

Malathi Veeraraghavan

T-Moors Date 18dec 2003

Tim Moors